Trabalho Final - Internet das Coisas

Rômulo Augusto Vieira Costa[1], **Rodrigo Dracxler**[1],
Carlos Salvador Gomide Richa[1], **Anderson Beltrão**[1]

[1]Instituto de Computação – Universidade Federal Fluminense (UFF)
R. Passo da Pátria, 156 - São Domingos, 24210-240
Niterói – RJ – Brasil

1. Introdução

O Brasil é um dos maiores produtores e consumidores de cerveja no mundo. Estima-se que 13,8 bilhões de litros sejam fabricados por ano no país, número que o coloca em terceiro lugar no *ranking* global, atrás apenas de China e Estados Unidos. Na última década, o consumo aumentou 5% ao ano, graças principalmente às cervejarias artesanais, que experimentam um crescimento paulatino desde o começo do século e, segundo a Associação Brasileira de Microcervejarias (Abracerva), somam mais de 400 estabelecimentos em todo o país [Vasconcelos 2021].

Para o ano de 2022 é esperado um crescimento sólido no mercado brasileiro de cerveja, em especial por conta do recrudescimento da pandemia e retorno de eventos sociais, como shows e festas, além da Copa do Mundo de futebol [Vasconcelos 2021, Forbes 2021].

Diante de um mercado tão robusto, várias iniciativas inovadoras são fomentadas em universidades, cervejarias, institutos de pesquisa e associações de agricultores, de modo a melhorar o processo produtivo de tal bebida, além de diminuir o impacto térmico nos produtos, reduzir o gasto de energia e o desperdício na produção, melhorar o envase, dentre outros. Das tecnologias que podem auxiliar este processo, destacam-se: inteligência artificial, genética, biotecnologia, impressão 3D e especialmente Internet das Coisas (IoT) [da Cerveja 2019, Prestes and Cordeiro 2008].

Para este trabalho, o foco será justamente na questão do impacto térmico na bebida. A ideia inicial era utilizar os dados gerados por dispositivos iSpindel[1] presentes na cadeia de produção de uma cervejaria artesanal real. Mas por conta deste equipamento possuir patente e utilizar código fechado, observou-se que isso não seria possível. Sendo assim, os autores simularam três sensores de temperatura acoplados em fermentadores, de modo a detectar mudanças nessa variável e ativar atuadores de alertas. Este ambiente é ilustrado na Figura 1, onde os atuadores são representados como lâmpadas, que acendem a partir de certos intervalos de temperatura detectados.

O restante do trabalho está dividido da seguinte maneira. A Seção 2 apresenta uma breve descrição das ferramentas utilizadas, enquanto a Seção

[1]https://github.com/universam1/iSpindel

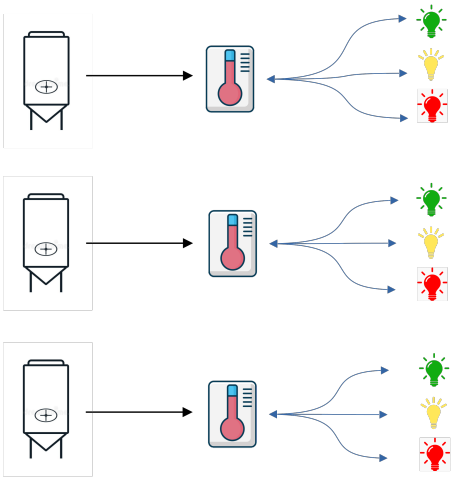

Figura 1. Desenho esquemático da simulação feita para o trabalho.
Fonte: O autor

3 foca na arquitetura e na relação de seus elementos. A Seção 4 mostra a implementação prática deste ambiente. A Seção 5 é encarregada de exibir os requisitos funcionais e não funcionais presentes no ambiente. Finalmente, a Seção 6 apresenta conclusões resumidas sobre a realização do trabalho, focando nas dificuldades enfrentadas e lições aprendidas.

2. Ferramentas utilizadas

A presente seção aborda as principais tecnologias utilizadas neste trabalho, como o Fiware, Orion Context Broker e IoT Agents (IDAS), usados para aquisição, processamento e atuação nos dados. Estes elementos serão melhores explicados a seguir.

2.1. Docker

O Docker é um conjunto de produtos que utiliza conteinerização de sistemas para facilitar o desenvolvimento dos mais variados tipos de serviços. Os contêineres gerados por esta aplicação são isolados um dos outros e agrupam seus próprios recursos, como bibliotecas e arquivos de configuração. Como eles são executados por um único núcleo (*kernel*) do sistema operacional, este paradigma aparece como uma alternativa às máquinas virtuais para realizar virtualização de serviços, já que são mais fáceis de configurar, apresentam menor perda de desempenho e tornam as operações mais intercambiáveis, eficientes e flexíveis. Além disso, elimina os problemas de dependência e compilação. Sua arquitetura básica pode ser observada na Figura 2 [Turnbull 2014, Moll 2022]. Note que ela apresenta em seu nível mais baixo a infraestrutura responsável por suportar o sistema, além da necessidade de um único sistema operacional para comportar todos os serviços necessários para a conteinerização. Por fim, a *engine* do sistema fica a cargo de gerar

diversas aplicações isoladas uma das outras, com suas próprias dependências e bibliotecas. Eventualmente, um contêiner pode ser interconectado a outro por meio de uma rede [Bashari Rad et al. 2017].

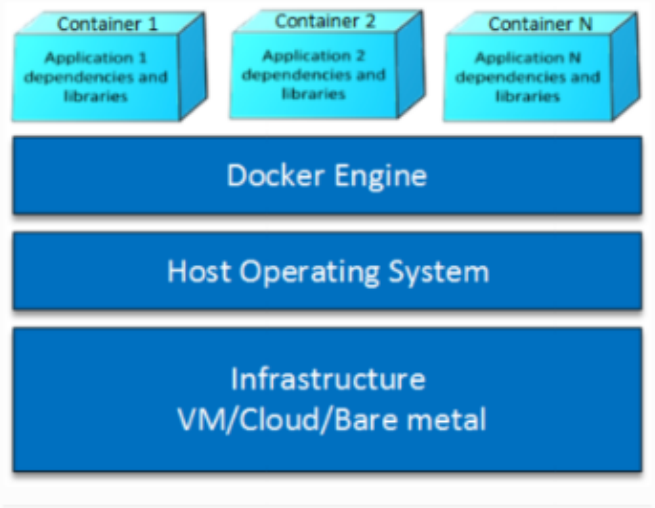

Figura 2. Arquitetura do Docker.
Fonte: [Moll 2022]

2.1.1. Imagens Docker

Como visto, cada contêiner é um ambiente isolado e contém um conjunto de processos que são executados a partir de uma Imagem Docker. Essas Imagens são consideradas o ponto inicial da arquitetura, pois através de suas compilações são gerados, de forma automática, os contêineres com o código da aplicação, bibliotecas, ferramentas, dependências e outros arquivos necessários para a execução de determinado serviço. Em outras palavras, servem como um *template* para ambiente (como o *snapshot* em uma máquina virtual) [Moll 2022].

Tradicionalmente, apresentam diversas camadas, onde cada nova camada é gerada a partir da camada imediatamente anterior, ainda que sejam substancialmente diferentes. Estas camadas agilizam a construção da arquitetura Docker e também facilitam sua reusabilidade e diminuem a quantidade de memória utilizada.

2.2. Fiware

O Fiware (Fig. 3) é uma plataforma (ou um *framework*) que contém diversos componentes de código aberto que podem ser usados com serviços de terceiros para implementar projetos inteligentes. Sua tecnologia visa a combinação da Internet das Coisas com Gerenciamento de Informações de Contexto e serviços de *Big Data* na nuvem para facilitar o processamento, análise e

visualização de informações. Isso permite que a plataforma seja usada na implementação de soluções bem elaboradas de cidades inteligentes, agricultura inteligente e indústria inteligente, que será o caso neste trabalho [Silva 2019].

Figura 3. Componentes do Fiware.
Fonte: [Silva 2019]

2.3. Orion Context Broker

Antes de mais nada, é importante entender o que é um *broker*. Ele pode ser definido como um servidor que recebe mensagens de seus clientes geradores de dados, conhecidos como **publicadores**, e repassa-as aos seus clientes consumidores, conhecidos como **inscritos**, atuando como uma espécie de mediador entre ambos, capaz de fazer com que a comunicação de fato ocorra. Além disso, faz a autenticação de dispositivos com base nas informações de conexão compartilhadas por eles, faz uso de criptografia e armazena mensagens no servidor para que possam ser reenviadas em caso de perda de conexão [Guner et al. 2017, Sen and Balasubramanian 2018].

Dentre as vantagens de seu uso básico, destacam-se: desacoplamento entre as partes; armazenamento e filtragem de informações, permitindo que cada dispositivo só receba as mensagens nas quais está interessado, reduzindo a largura de banda necessária para essa comunicação; troca de dados ativa e simultânea com diversos *gateways* e dispositivos IoT para obter informações exigidas pelas aplicações; escalabilidade; separação das aplicações IoT das implementações de dispositivos subjacentes; informações geradas e trocadas somente quando necessário e tradução automática de informações para o nível de abstração correto [Guner et al. 2017].

Um *broker* pode ser tanto um servidor local quanto uma estrutura em nuvem. Alguns exemplos são: ActiveMQ, RabbitMQ, Mosquitto, ZeroMQ e Orion Context Broker. Para que uma aplicação criada por um desenvolvedor

seja considerada “Powered by FIWARE”, é obrigatório o uso do Orion Context Broker, uma vez que ele traz a funcionalidade que é conhecida como pilar do Fiware: o gerenciamento de informações de contexto.

O funcionamento básico do Orion Context Broker consiste em consultar no Consumidor de Contexto as informações fornecidas pelo Produtor de Contexto. Ele é utilizado para criar as entidades de contexto, ao passo que o Produtor de Contexto atualiza as entidades e atributos da aplicação específica e o Consumidor consulta as informações de interesse, tendo a possibilidade de subscrever-se às mudanças de contexto para ser notificado assincronamente a cada vez que uma informação é modificada [Silva 2019].

2.4. IoT Agents

A Internet das Coisas apresenta diversas definições, mas todas elas concordam que tal campo de pesquisa trata de uma rede de dispositivos capazes de se conectarem e trocarem dados para atingirem um objetivo em comum. Esses dispositivos são chamados de “coisas” e nada mais são que *gadgets* embarcados com sistemas eletrônicos e capacidade computacional, capazes de atuar como sensores ou atuadores. Estes sensores são capazes de reportar o estado de um evento do mundo real, enquanto os atuadores podem responder a um sinal de controle e alterar o estado de um sistema [Atzori et al. 2010, Gershenfeld et al. 2004].

Apesar das diversas vantagens de se utilizar estes dispositivos, eles se deparam com problemas de padronização de dados e protocolos, o que dificulta a interoperabilidade entre eles. Uma alternativa para isso é o uso de IoT Agents - IDAS (Fig. 4), que tem o propósito facilitar a interação de dispositivos com outros sistemas para coleta de informações de contexto ou para disparar eventos caso ocorra uma mudança nas variáveis [Silva 2019].

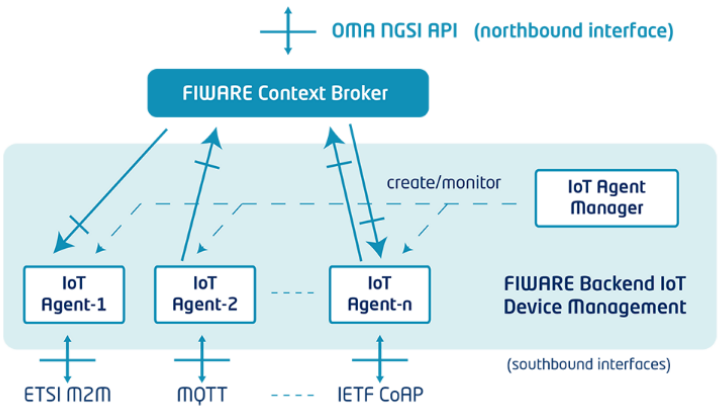

Figura 4. Arquitetura básica de uma aplicação que utiliza o IDAS para conectar dispositivos inteligentes ao Orion Context Broker.
Fonte: [Silva 2019]

Segundo o catálogo do IDAS, estão disponíveis os seguintes modelos:

- **IoT Agent para JSON;**
- **IoT Agent para LWM2M;**
- **IoT Agent para Ultralight;**
- **IoT Agent para LoRaWAN;**
- **IoT Agent para OPC-UA;**
- **IoT Agent para Sigfox.**

O projeto em questão irá usar o UltraLight, na versão 2.0. Este protocolo é baseado em texto e é utilizado em dispositivos restritos, onde a capacidade de largura de banda e de memória são limitadas.

O UltraLight define uma carga útil de trabalho capaz de descrever medidas e comandos para compartilhar informações entre dispositivos e servidores, mas sem se pautar por um único protocolo de transporte, utilizando desde o HTTP (Hypertext Transfer Protocol) até o MQTT (Message Queuing Telemetry Transport) e o AMQP (Advanced Message Queuing Protocol). Esta variedade permite que ele seja aplicável aos mais diferentes cenários de uso.

2.5. MongoDB

O MongoDB [Kovacs 2021] é um banco de dados NoSQL de código aberto, orientado para documentos no formato JSON, ou seja, os dados são armazenados como documentos, ao contrário de um banco de dados relacional, onde são usados registros em linhas e colunas. Foi projetado para permitir o trabalho de forma ágil e eficiente em grandes volumes de dados. Dentre suas principais características, destacam-se:

- **Sintaxe para consultas:** Capaz de lidar com consultas que vão desde as mais básicas até buscas mais complexas. Também permite consultas por meio do JavaScript;
- **Indexação:** Conceito similar àquele presente nos bancos de dados relacionais, permitindo a criação de índices que impactam no desempenho de suas consultas a medida que o banco de dados aumenta de tamanho;
- **Escalabilidade horizontal:** Conceito referente a necessidade do banco de dados ser adaptável, aumentando seu tamanho em função das informações que são armazenadas pelos usuários. A escalabilidade horizontal envolve a divisão do conjunto de dados do sistema e a carga em vários servidores, capazes de aumentar a capacidade e disponibilidade da aplicação, de acordo com o volume dos dados ou o número de acessos ao banco.

Quanto às suas vantagens, pode-se citar flexibilidade, disponibilidade e alto desempenho. Também apresenta tolerância automática a falhas e oferece confiabilidade ao projeto.

2.6. cURL

O Client Uniform Resource Locator, mais conhecido por cURL, é uma ferramenta de linha de comando criada por Daniel Stenberg, com o objetivo de

desenvolver um *bot* para converter taxas de câmbio de uma página web para valores em dólar. Posteriormente, evoluiu para um meio capaz de transferir dados de/para um servidor usando uma ampla gama de protocolos, em especial o HTTP.

Para além disso, também é capaz de fazer o *download* de documentos, reiniciar estes mesmos *downloads* quando interrompidos, realizar consultas em cabeçalhos HTTP, *download* ou *upload* de arquivos em um servidor File Transfer Protocol (FTP) com ou sem autenticação, descobrir os *cookies* armazenados durante um processo e etc.

3. Arquitetura

A medição da temperatura da cerveja é um fator-chave no processo de fermentação, uma vez que ela é capaz de alterar os valores alcoólicos da bebida, o que afeta diretamente o resultado final do produto. A arquitetura básica do sistema responsável por realizar este controle pode ser observada na Figura 5. Repare que o sensor de temperatura se comunica com o Orion Context Broker, responsável pelas atualizações de contexto, através do IoT Agent. O atuador, por sua vez, irá receber essas medições ambém por meio do IDAS, para a partir da lógica de negócio implementada, disparar sinais luminosos de acordo com as seguintes condições:

- Se temperatura for menor 40º C, acende a luz verde;
- Se a temperatura estiver entre 41º C e 60º C, atua a luz amarela;
- Se a temperatura for maior que 60º C, acende a luz vermelha.

3.1. Diagrama UML

A Unified Modeling Language (UML)[2][3] foi criada por Grady Booch, Ivar Jacobson e James Rumbaugh no ano de 1994 para modelar soluções de *software*, estruturas de aplicações, comportamentos de sistemas e processos de negócio. Em termos de *design*, oferece um meio de visualizar a arquitetura de um sistema por meio de diversos diagramas, incluindo atividades, componentes do sistema e as interações desses componentes entre si e também com o mundo externo. Vale destacar que essa forma de modelagem não é um método de desenvolvimento, plataforma ou linguagem de programação, mas sim um padrão para descrever **o que** e c**omo fazer** [Booch et al. 2000].

Atualmente, existem 14 tipos de diagramas UML divididos em duas categorias: diagramas estruturais e diagramas comportamentais, sintetizados na Figura 6[4]. Sua aplicação favorece todo o *stakeholder* envolvido em um processo, indo deste os proprietários dos produtos e analistas de negócios até os operadores do sistema, desenvolvedores e gerentes de qualidade.

[2]https://www.uml.org/

[3]https://www.devmedia.com.br/o-que-e-uml-e-diagramas-de-caso-de-uso-introducao-pratica-a-uml/23408

[4]https://www.lucidchart.com/pages/pt/o-que-e-uml

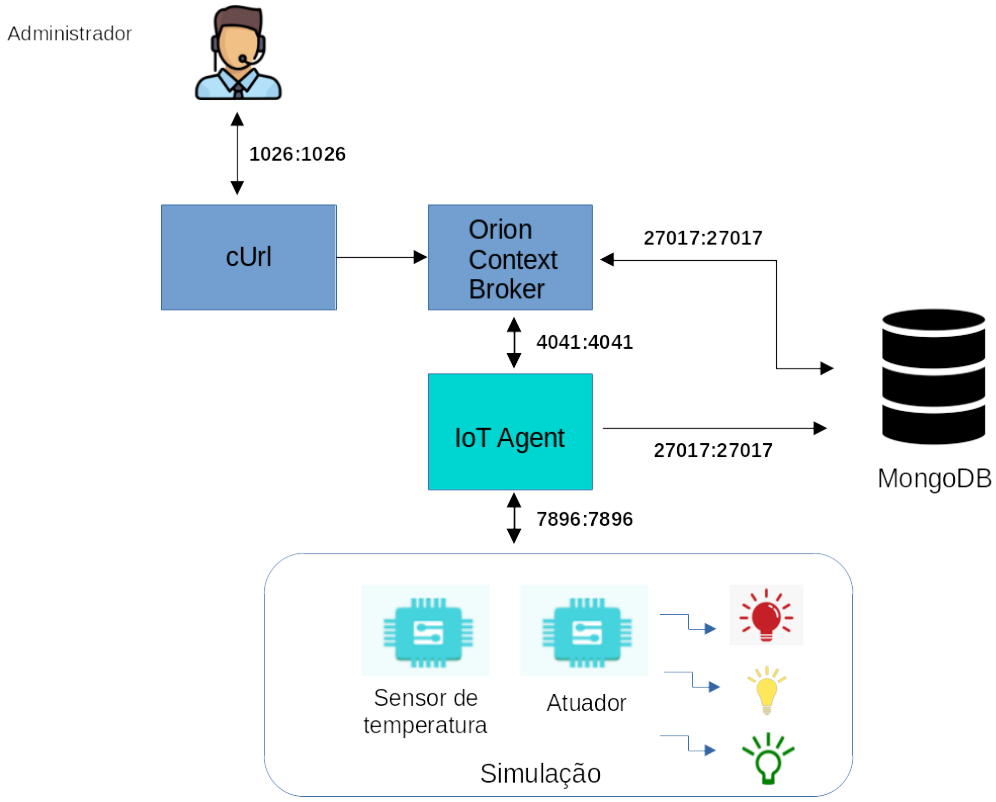

Figura 5. Arquitetura do projeto.
Fonte: O autor

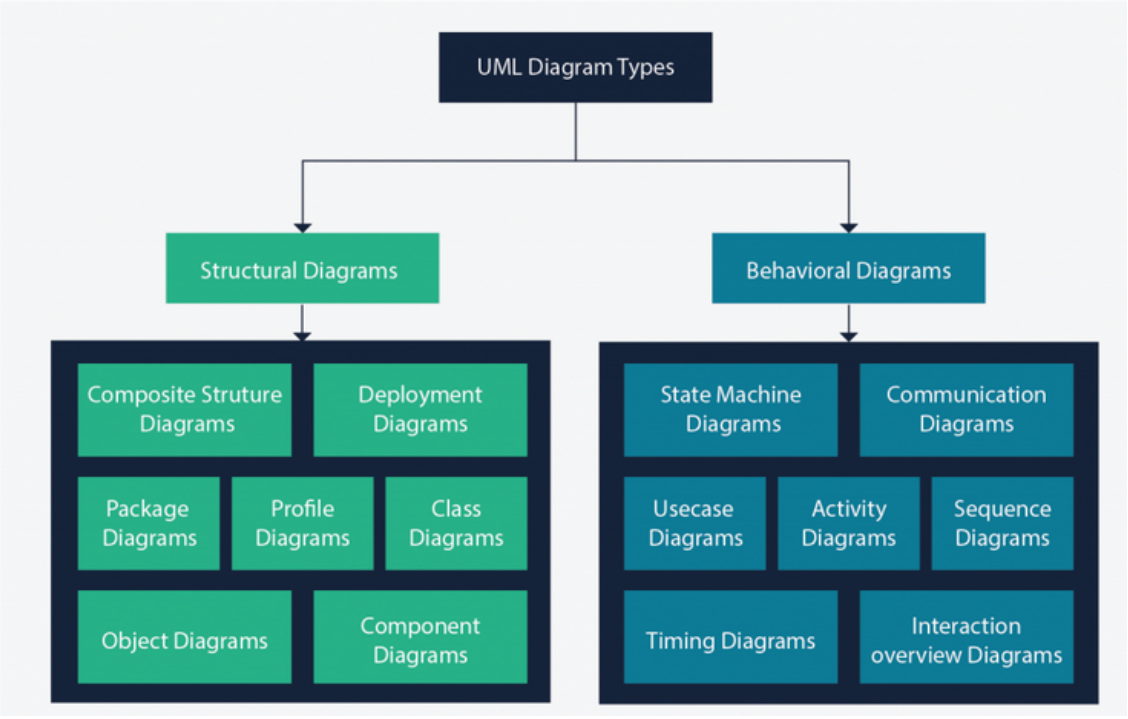

Figura 6. Os 14 tipos de diagramas UML.
Fonte: Lucid Chart

Para o trabalho, foi elaborado um **diagrama de objetos**, também conhecido como diagrama de instâncias. Ele foca na relação entre objetos em um momento específico e nos dados disponíveis em cada um deles. É útil para examinar de forma específica cada interação do sistema, bem como obter uma visão de alto-nível sobre ele.

Um objeto deve conter quatro parâmetros: instância ou nome do objeto, nome da classe, atributo e valor. A Figura 7 exemplifica onde devem estar estes atributos no bloco criado.

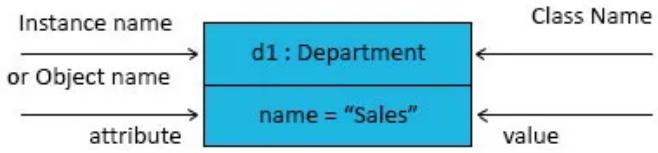

Figura 7. Representação esquemática de um objeto no diagrama UML.
Fonte: EDUCBA

Já a interconexão de objetos pode acontecer por meio de ***links***, utilizado para representar a relação entre as instâncias; **associação**, que retrata a dependência de um objeto por outro; **composição**, quando um objeto não pode ocorrer sem a presença de outro objeto, e **agregação**, que denota relação sem dependência entre objetos. A Figura 8 mostra a representação gráfica destes conectores.

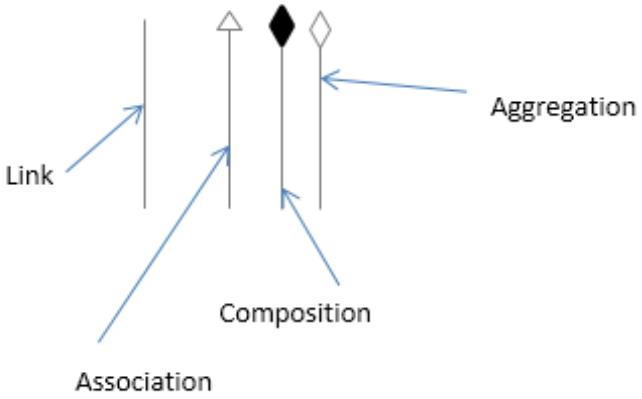

Figura 8. Conectores usados em um diagrama UML.
Fonte: EDUCBA

Na Figura 9 é possível visualizar o Diagrama UML para o trabalho proposto. Repare nas especificações de cada elemento, como *tags* e valores para os sensores, além da relação deles com o IoT Agents e com o Orion Context Broker.

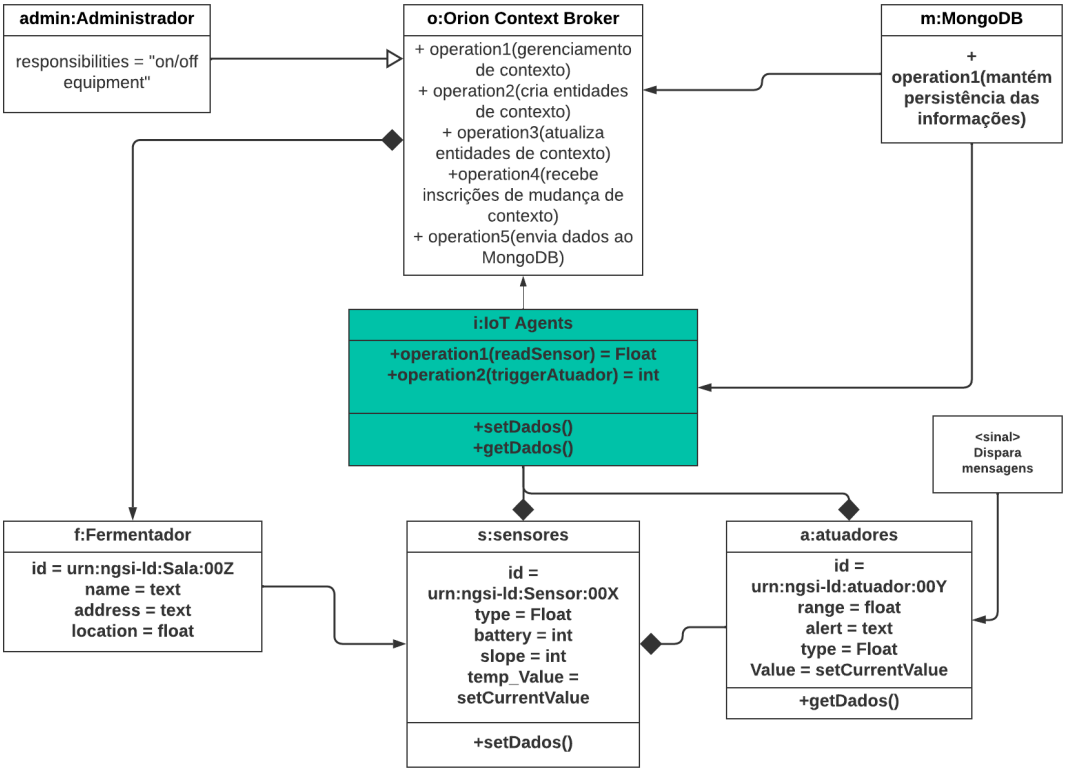

Figura 9. Diagrama UML para o trabalho final da disciplina de IoT.
Fonte: O Autor

4. Aplicação

A implementação do trabalho se vale dos conceitos da Internet das Coisas e de todas as tecnologias por trás do FIWARE, de modo a colocar em prática os conceitos e ferramentas aprendidos em aula. A seguir, está listado um passo a passo para criar o ambiente e executar algumas funções nele. Ele foi executado em um Linux Ubuntu Pop_OS 20.04 e os arquivos e códigos necessários encontram-se no GitHub[5]. Ainda, um vídeo-demonstração do trabalho pode ser encontrado no YouTube[6].

a) Instale as ferramentas necessárias

- Instale o Docker[7];
- Instale o Docker compose (necessário apenas para ambientes UNIX)[8];
- Instale o MongoDB[9];
- Instale o cURL[10].

[5]https://github.com/romulovieira-me/trabalho-iot
[6]https://youtu.be/TAbDXc99gqk
[7]https://docs.docker.com/engine/install/ubuntu/docker compose docker
[8]https://docs.docker.com/compose/install/mongo
[9]https://www.mongodb.com/docs/manual/administration/install-community/
[10]https://www.cyberciti.biz/faq/how-to-install-curl-command-on-a-ubuntu-linux/

b) Inicie o ambiente

- Crie um arquivo YML com o seguinte contéudo

```
# Trabalho final de Internet das Coisas
# Grupo: Romulo Vieira, Rodrigo Dracxler, Anderson Beltrao, Carlos Salvador

version: "3.1"
# Declarando servicos
services:
# Declarando mongo
    mongo-db:
        image: mongo:4.4 # versao da imagem do mongo
        hostname: mongo-db # hostname para acessa-lo
        container_name: banco-mongodb # nome do container
        expose:
            - "27017" # porta para acessar o banco de dados
        ports:
            - "27017:27017" # portas de entrada e saida para o banco de dados
        networks: # criando servico de rede para interligar conteiners
            - default
        command: --bind_ip_all --smallfiles
        volumes:
            - mongo-db:/data
    orion: # criando servico do context orion broker
        image: fiware/orion # imagem do Orion baixada do repositorio docker
        hostname: orion
        container_name: fiware-orion-trabalho-iot
        depends_on: # criando dependencia
            - mongo-db
        networks:
            - default
        expose:
            - "1026" # portas do orion context broker
            - "3000"
            - "3001"
        ports:
            - "1077:1026"
            - "3000:3000"
        command: -dbhost mongo-db -logLevel DEBUG
        healthcheck:
```

```
38          test: curl --fail -s http://localhost:1026/version ||
               exit 1 # comando para testar se context broker esta
               funcionando
39
40      iot-agent: # criando servico do IoT Agent
41          image: fiware/iotagent-ul:1.8.0
42          hostname: iot-agent
43          container_name: fiware-iot-agent-trabalho
44          depends_on:
45              - mongo-db
46          networks:
47              - default
48          expose:
49              - "4041"
50              - "7896"
51          ports:
52              - "4041:4041"
53              - "7896:7896"
54          environment: # especificacoes do IoT Agent
55              - "IOTA_CB_HOST=orion" # hostname do context broker
                   para atualizar contexto
56              - "IOTA_CB_PORT=1026" # Porta que o context broker
                   escuta para atualizar contexto
57              - "IOTA_NORTH_PORT=4041" # Porta usada para
                   configurar o IoT Agent e receber atualizacao de
                   contexto do broker
58              - "IOTA_REGISTRY_TYPE=mongodb" # instancia do mongoDB
59              - "IOTA_LOG_LEVEL=DEBUG" # opcao de debug para o IoT
                   Agent
60              - "IOTA_TIMESTAMP=true"
61              - "IOTA_MONGO_HOST=mongo-db" # hostname do mongoDB -
                   usado para manter informacoes dos dispositivos
62              - "IOTA_MONGO_PORT=27017" # porta onde o mongoDB esta
                    ouvindo
63              - "IOTA_MONGO_DB=iotagentul" # nome da base de dados
                   usada pelo mongoDB
64              - "IOTA_HTTP_PORT=7896" # porta onde o IoT Agent ouve
                    o trafego dos dispositivos IoT
65              - "IOTA_PROVIDER_URL=http://iot-agent:4041" # URL
                   passado ao context broker quando os comandos sao
                   registrados
66          healthcheck:
67              test: curl --fail -s http://localhost:4041/iot/about
                   || exit 1 # testando o funcionamento da IoT Agent
```

```
# Encerrando servicos
networks:
  default:
    ipam:
        config:
          - subnet: 172.10.1.0/24

volumes:
  mongo-db:
```

- Execute o arquivo com o seguinte comando

```
    docker-compose --log-level ERROR -p fiware up -d --remove-
       orphans
```

c) Torne o banco de dados persistente

```

    docker run --name bancodadosmongodb -v /home/romulo/mongodata
       :/data/db -d mongo
```

d) Verifique se o Orion Context Broker está funcionando

```

    curl -X GET \
  'http://localhost:1026/version'
```

e) Crie dados de contexto

Para iniciar os elementos que fornecerão dados ao nosso ambiente, crie três fermentadores, três sensores de temperatura e três atuadores. É possível criar tudo através de uma única requisição, mas como aqui é um tutorial, tudo será feito passo a passo.

f) Crie o Fermentador 1

```

   curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -d '
{
    "id": "urn:ngsi-ld:Fermentador:101",
    "type": "Fermentador",
    "address": {
        "type": "PostalAddress",
```

```
        "value": {
            "streetAddress": "Rua Jose Saramago 25",
            "addressRegion": "Rio de Janeiro",
            "postalCode": "10439"
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [13.3986, 52.5547]
        }
    },
    "temperature": {
        "type": "Float",
        "value": "24"
    }
}'
```

g) Crie o Fermentador 2

```

   curl -iX POST \
  'http://localhost:1026/v2/entities' \
  -H 'Content-Type: application/json' \
  -d '
{
    "id": "urn:ngsi-ld:Fermentador:102",
    "type": "Fermentador",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Rua Jose Saramago 25",
            "addressRegion": "Rio de Janeiro",
            "postalCode": "10439"
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [13.3986, 52.5547]
        }
    },
    "temperature": {
```

```
        "type": "Float",
        "value": "45"
    }
}’
```

h) Crie o Fermentador 3

```

   curl -iX POST \
  ’http://localhost:1026/v2/entities’ \
  -H ’Content-Type: application/json’ \
  -d ’
{
    "id": "urn:ngsi-ld:Fermentador:103",
    "type": "Fermentador",
    "address": {
        "type": "PostalAddress",
        "value": {
            "streetAddress": "Rua Jose Saramago 25",
            "addressRegion": "Rio de Janeiro",
            "postalCode": "10439"
        }
    },
    "location": {
        "type": "geo:json",
        "value": {
             "type": "Point",
             "coordinates": [13.3986, 52.5547]
        }
    },
    "temperature": {
        "type": "Float",
        "value": "78.13"
    }
}’
```

i) Crie Grupo de serviço

Criar grupos é o primeiro passo para conectar dispositivos, uma vez que é necessário fornecer uma chave de autenticação (que no código abaixo é: 4jggokgpepnvsb2uv4s40d59ov) para alcançar os aparelhos. Além de informar ao IoT Agent em qual URL o Orion Context Broker está respondendo.

```

  curl -iX POST \
  ’http://localhost:4041/iot/services’ \
```

```
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "services": [
   {
     "apikey":      "4jggokgpepnvsb2uv4s40d59ov",
     "cbroker":     "http://orion:1077",
     "entity_type": "Thing",
     "resource":    "/iot/d"
   }
 ]
}'
```

j) Crie Sensor 1

```
 curl -iX POST \
 'http://localhost:4041/iot/devices' \
 -H 'Content-Type: application/json' \
 -H 'fiware-service: openiot' \
 -H 'fiware-servicepath: /' \
 -d '{
"devices": [
  {
    "device_id":   "temperatura001",
    "entity_name": "urn:ngsi-ld:Temperatura:001",
    "entity_type": "Temperatura",
    "timezone":    "America/Rio de Janeiro",
    "attributes": [
      { "object_id": "c", "name": "count", "type": "Float" }
    ],
    "static_attributes": [
      { "name":"refFermentador", "type": "Relationship", "value"
         : "urn:ngsi-ld:Fermentador:101"}
    ]
  }
 ]
}
'
```

k) Crie Sensor 2

```
 curl -iX POST \
```

```
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
   {
     "device_id":   "temperatura002",
     "entity_name": "urn:ngsi-ld:Temperatura:002",
     "entity_type": "Temperatura",
     "timezone":    "America/Rio de Janeiro",
     "attributes": [
       { "object_id": "c", "name": "count", "type": "Float" }
     ],
     "static_attributes": [
       { "name":"refFermentador", "type": "Relationship", "value"
          : "urn:ngsi-ld:Fermentador:102"}
     ]
   }
 ]
}
'
```

l) Crie Sensor 3

```

  curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
 "devices": [
   {
     "device_id":   "temperatura003",
     "entity_name": "urn:ngsi-ld:Temperatura:003",
     "entity_type": "Temperatura",
     "timezone":    "America/Rio de Janeiro",
     "attributes": [
       { "object_id": "c", "name": "count", "type": "Float" }
     ],
     "static_attributes": [
       { "name":"refFermentador", "type": "Relationship", "value"
          : "urn:ngsi-ld:Fermentador:003"}
     ]
```

```
    }
  ]
}
'
```

o) Simule dados para o Sensor 1

```

curl -iX POST \
  'http://localhost:7896/iot/json?k=4jggokgpepnvsb2uv4s40d59ov&i=
     Temperatura001' \
  -H 'Content-Type: application/json' \
  -d 'c|24'
```

p) Simule dados para o Sensor 2

```

curl -iX POST \
  'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=
     temperatura002' \
  -H 'Content-Type: text/plain' \
  -d 'c|45'
```

q) Simule dados para o Sensor 3

```

curl -iX POST \
  'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=
     temperatura003' \
  -H 'Content-Type: text/plain' \
-d 'c|77'
```

r) Recupere informações do Orion Context Broker

```

curl -iX POST \
  'http://localhost:7896/iot/d?k=4jggokgpepnvsb2uv4s40d59ov&i=
     temperatura002' \
  -H 'Content-Type: text/plain' \
  -d 'c|45'
```

s) Crie um atuador 1

O processo de criação de um atuador é semelhante ao de um sensor, com a diferença que seu endereço final será a localização onde o IoT Agent precisa enviar os dados.

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [
    {
      "device_id": "atuador001",
      "entity_name": "urn:ngsi-ld:Atuador:001",
      "entity_type": "Atuador",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://iot-sensors:3001/iot/Atuador001",
      "commands": [
        {"name": "on","type": "command"},
        {"name": "off","type": "command"}
       ],
       "attributes": [
        {"object_id": "s", "name": "state", "type":"Text"},
        {"object_id": "l", "name": "luminosity", "type":"Integer"
           }
       ],
       "static_attributes": [
         {"name":"refTemperatura", "type": "Relationship","value"
            : "urn:ngsi-ld:Temperatura:001"}
      ]
    }
  ]
}
'
```

t) Crie um atuador 2

```
curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
```

```
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [
    {
      "device_id": "Atuador002",
      "entity_name": "urn:ngsi-ld:Atuador:002",
      "entity_type": "Atuador",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://iot-sensors:3010/iot/Atuador002",
      "commands": [
        {"name": "on","type": "command"},
        {"name": "off","type": "command"}
       ],
       "attributes": [
        {"object_id": "s", "name": "state", "type":"Text"},
        {"object_id": "l", "name": "luminosity", "type":"Integer"
           }
       ],
       "static_attributes": [
         {"name":"refTemperatura", "type": "Relationship","value"
            : "urn:ngsi-ld:Temperatura:002"}
      ]
    }
  ]
}
'
```

u) Crie um atuador 3

```

curl -iX POST \
  'http://localhost:4041/iot/devices' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "devices": [
    {
      "device_id": "Atuador003",
      "entity_name": "urn:ngsi-ld:Atuador:003",
      "entity_type": "Atuador",
      "protocol": "PDI-IoTA-UltraLight",
      "transport": "HTTP",
      "endpoint": "http://iot-sensors:3010/iot/Atuador003",
```

```
      "commands": [
        {"name": "on","type": "command"},
        {"name": "off","type": "command"}
       ],
       "attributes": [
        {"object_id": "s", "name": "state", "type":"Text"},
        {"object_id": "l", "name": "luminosity", "type":"Integer"
           }
       ],
       "static_attributes": [
         {"name":"refTemperatura", "type": "Relationship","value"
            : "urn:ngsi-ld:Temperatura:003"}
      ]
    }
  ]
}
'
```

v) Teste o envio de comando para o atuador

```

curl -X GET \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Atuador:001?type
     =Atuador&options=keyValues' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

w) Acione o atuador

```

curl -iX PATCH \
  'http://localhost:1026/v2/entities/urn:ngsi-ld:Atuador:001/
     attrs' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "on": {
      "type" : "command",
      "value" : ""
  }
}'
```

x) Crie instruções para o atuador 1 receber alertas caso haja mudanças no contexto do sensor de temperatura 1

```

curl -iX POST \
  --url 'http://localhost:1026/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Subscription do atuador 1",
  "subject": {
    "entities": [{"id": "urn:ngsi-ld:Atuador001", "type": "
       Atuador"}],
    "condition": {
      "attrs": [ "count" ]
    }
  },
  "notification": {
    "http": {
      "url": "http://localhost:3000/subscription/mudanca-
         temperatura"
    }
  }
}''
```

y) Crie instruções para o atuador 2 receber alertas caso haja mudanças no contexto do sensor de temperatura 2

```

curl -iX POST \
  --url 'http://localhost:1077/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Subscription do atuador 2",
  "subject": {
    "entities": [{"id": "urn:ngsi-ld:Atuador:002", "type": "
       Atuador"}],
    "condition": {
      "attrs": [ "count" ]
      "expression": ["q": "count>40&&count<60; refTemperatura==
         urn:ngsi-ld:Temperatura:002" ]
 }
  },
  "notification": {
    "http": {
      "url": "http://tutorial:1077/subscription/mudanca-
         temperatura"
```

```
    }
  }
}'
```

z) Crie instruções para o atuador 3 receber alertas caso haja mudanças no contexto do sensor de temperatura 3

```

curl -iX POST \
  --url 'http://localhost:1077/v2/subscriptions' \
  --header 'content-type: application/json' \
  --data '{
  "description": "Subscription do atuador 3",
  "subject": {
    "entities": [{"id": "urn:ngsi-ld:Atuador:003", "type": "
       Atuador"}],
    "condition": {
      "attrs": [ "count" ]
      "expression": ["q": "count>60; refTemperatura==urn:ngsi-ld:
         Temperatura:003" ]
    }
  },
  "notification": {
    "http": {
      "url": "http://tutorial:1077/subscription/mudanca-
         temperatura"
    }
  }
}'
```

a2) Delete uma inscrição

```

curl -X DELETE \
  --url 'http://localhost:1026/v2/subscriptions/
```

b2) Obtendo lista de todas as inscrições

```

curl -X GET \
  --url 'http://localhost:1026/v2/subscriptions'
```

4.1. Comandos Extras

Para expandir ainda mais as capacidades de controle sobre o ambiente, são listadas mais algumas tarefas a seguir.

Obtendo todos os dispositivos presentes

```
curl -X GET \
  'http://localhost:4041/iot/devices' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

Atualizando um dispositivo

```
curl -iX PUT \
  'http://localhost:4041/iot/devices/Atuador001' \
  -H 'Content-Type: application/json' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /' \
  -d '{
  "entity_type": "IoT-Device"
}'
```

Deletando um dispositivo

```
curl -iX DELETE \
  'http://localhost:4041/iot/devices/Atuador002' \
  -H 'fiware-service: openiot' \
  -H 'fiware-servicepath: /'
```

Obtendo dados da entidade pela ID

```
curl -X GET \
   'http://localhost:1026/v2/entities/urn:ngsi-ld:Fermentador:001
     ?options=keyValues'
```

Obtendo dados da entidade pelo tipo

```
curl -X GET \
    'http://localhost:1026/v2/entities?type=Fermentador&options=
      keyValues'
```

Lendo dados a partir do ID

```
curl -X GET \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:
    Temperatura:001?type=Temperatura'
```

Lendo entidades a partir de seu nome

```
curl -X GET \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:
     Temperatura:001/attrs/name/value'
```

Listando todos os dados de entrada

```
curl -X GET \
  --url 'http://localhost:1026/v2/entities?type=Temperatura'
```

Reescrevendo o valor de um atributo a partir de sua ID

```
curl -iX PUT \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:
     Temperatura:001/attrs/count/value' \
  --header 'Content-Type: text/plain' \
  --data 89
```

Substituindo um dado

```
curl -iX POST \
  --url 'http://localhost:1026/v2/op/update' \
  --header 'Content-Type: application/json' \
  --data '{
  "actionType":"replace",
  "entities":[
    {
      "id":"urn:ngsi-ld:Temperatura:001", "type":"Temperatura",
      "price":{"type":"Float", "value": 1199}
    }
  ]
}'
```

Deletando dados a partir da ID

```
curl -iX DELETE \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:
     Temperatura:001'
```

Deletando atributos

```
curl -iX DELETE \
  --url 'http://localhost:1026/v2/entities/urn:ngsi-ld:
     Temperatura:001/attrs/count'
```

5. Requisitos

As especificações de um sistema ou engenharia de requisitos é o processo de compreensão e definição dos serviços necessários para a realização de uma determinada tarefa, bem como a identificação de restrições relativas à operação e desenvolvimento do sistema [Sommerville 2019, Amaral et al. 2021].

Outra definição de requisitos bastante utilizada é o conceito da norma IEEE 610.12/1990, que descreve requisito como: a) uma condição ou capacidade requerida pelos usuários para resolver um problema ou atingir um objetivo; b) uma condição ou capacidade de um sistema para satisfazer um padrão, contrato ou especificação requerida; c) uma representação documentada da condição ou capacidade expressa em A e B. De forma geral, essas três definições concordam que requisitos devem abranger a visão do usuário (comportamento externo do sistema), a visão do desenvolvedor (comportamento interno do sistema) e o conceito fundamental de que os requisitos devem ser documentados [Pohl 2012, de Padua Paula Filho 2019, Vazquez and Simoes 2016].

Os requisitos podem ser classificados em funcionais, que indica o que o sistema deve ou não fazer em termos de tarefas e funcionalidades, e não-funcionais, que indicam as restrições técnicas, questões de confiabilidade, desempenho, tempo de resposta e outros itens que indicam como o sistema irá funcionar [Amaral et al. 2021].

5.1. Requisitos Funcionais

Os requisitos funcionais são intrinsecamente relacionados às demandas dos usuários do sistema, podendo ser requisitos gerais, expressos como requisitos de usuários, ou requisitos muito específicos, que refletem as formas de trabalho dentro de uma organização e, por isso, necessitam ser mais detalhados.

Para o presente trabalho, foram definidos os seguintes requisitos que cumprem essas especificações:

- **Gerenciamento de Recursos:** Entende-se como recurso qualquer objeto que pode ser alocado dentro de um sistema. E antes que possa ser usado, um recursos precisa ser descoberto. Como o sistema aqui apresentado simula uma aplicação cervejeira real, a rede é infraestruturada, de modo que os sensores que vão fornecer recursos ao ambiente são conhecidos de antemão, de forma que puderam ser mapeados de forma direta, não necessitando de um meio para sua descoberta.

- **Gerenciamento de Dados:** Dados são fundamentais em aplicações IoT, pois vão fornecer o contexto da aplicação. É importante, portanto, que haja o gerenciamento desse contexto, capaz de receber dados dos sensores, processá-los e tomar decisões com base nos mesmos;
- **Adaptação:** É comum que haja mudanças frequentes no ambiente por conta da mobilidade dos nós, dreno de energia, mudanças de topologia e etc. As constantes mudanças na temperatura exigem serviços de adaptação capazes de prover formas de reagir a essas alterações. Os atuadores devem ser sensíveis a isso, de modo que sejam notificados e atuem de maneira a responder a estas modificações.
- **Gerenciamento de Eventos:** Um grande número de eventos são gerados em aplicações IoT e esses eventos fornecem dados significativos para a aplicação. No ambiente simulado, os eventos mais significativos ocorrem nos fermentadores, com suas temperaturas sofrendo constantes variações. Cada uma dessas variações irá gerar eventos para os atuadores, que serão acionados em resposta.

5.2. Requisitos Não Funcionais

Os requisitos não-funcionais expressam propriedades e restrições do sistema, sendo, frequentemente, mais críticos que os requisitos funcionais. As determinações presentes neste ambiente podem ser vistas abaixo.

- **Escalabilidade:** Sistema deve ser capaz de acomodar o crescimento da rede, dos dados e das aplicações (como por exemplo, inserção de novos fermentadores, sensores e atuadores), bem como assimilar um número crescente de dispositivos e requisições/publicações, gerenciamento de endereços de dispositivos e gerenciamento de grandes volumes de dados.
- **Operar em tempo real:** Como as mais simples mudanças na temperatura podem alterar de forma drástica o resultado final do produto, é de vital importância que ele seja dependente do tempo de execução.
- **Interoperabilidade:** O sistema deve dar suporte e interoperar com dispositivos, tecnologias e aplicações heterogêneas sem requerer esforço adicional, além de apresentar múltiplos níveis de interoperabilidade (dispositivos e redes) e integração sintática e semântica.

Após todas essas elucubrações, a Tabela 1 sintetiza todos os requisitos funcionais e não funcionais para o sistema.

Tabela 1. Tabela de requisitos.

Requisitos Funcionais	Requisitos Não Funcionais
Gerenciamento de Recursos	Escalabilidade
Gerenciamento de Dados	Operar em tempo real
Adaptação	Interoperabilidade
Gerenciamento de Eventos	–

Fonte: O Autor.

6. Conclusão

O presente trabalho apresentou um ambiente industrial cervejeiro simulado, com o intuito de desenvolver aplicações de Internet das Coisas baseadas no Fiware e colocar em prática os conceitos aprendidos ao longo do semestre letivo, como ciência de contexto, *middleware* para IoT, protocolos de comunicação e requisitos de ambiente, sendo o objetivo principal entender como se dá a comunicação entre os componentes da aplicação e o Fiware (em particular, o Broker). Tal comportamento é resumido na Figura 10.

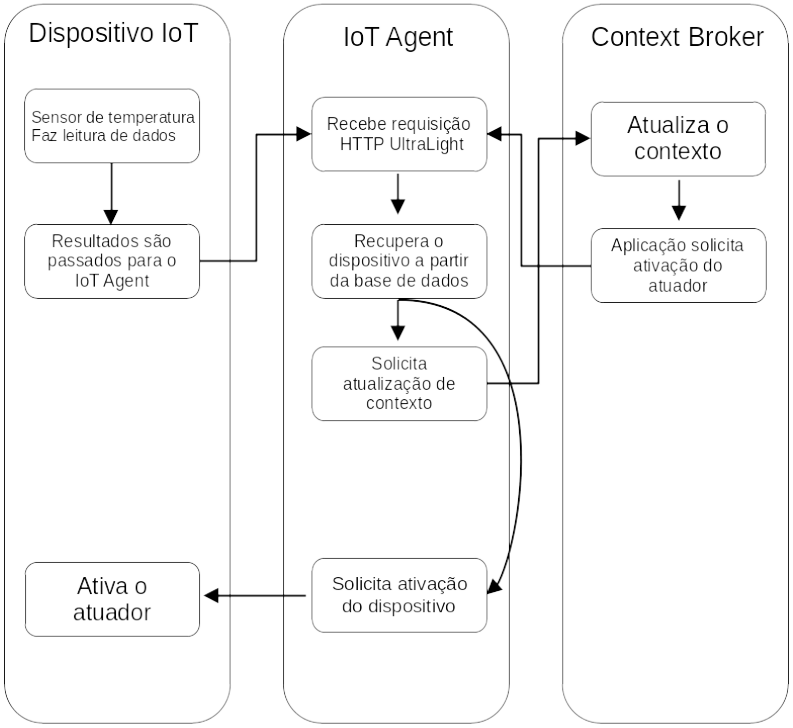

Figura 10. Comunicação entre o FIware e os elementos da aplicação.
Fonte: O autor

Como citado, a ideia inicial era colher dados de sensores reais implementados em fermentadores e enviar seus dados via Orion Context Broker para atuadores simulados. Entretanto, tais sensores eram de código fechado, o que inviabilizou tal atividade.

Diante disso, os autores recorreram às funcionalidades do IoT Agent e simularam os sensores por lá também. Os demais dados de contexto, como os fermentadores, também foram criados de forma virtual para a conclusão do trabalho.

Em um primeiro momento, os autores se depararam com alguma dificuldade no uso da plataforma Docker, em especial pelo fato dela manter os contêineres de forma persistente, o que causou alguns conflitos de *hostname*

e número de portas. O tutorial fragmentado, e por vezes desatualizado, do Orion Context Broker e a falta de trabalhos relacionados também foram desafios a serem superados.

Entretanto, após algum tempo de prática, todas as ferramentas foram aprendidas de modo que puderam ser exploradas e aplicadas neste ambiente simulado. O fato de boa parte desses recursos serem *backend* e não apresentarem interface gráfica não foi um problema, visto que o autores possuíam alguma experiência nesta área. O mesmo é válido para o banco de dados, cURL e para a linguagem JSON.

Alguma experiência e conhecimento na área de conteinerização e virtualização de serviços, sistemas operacionais (em especial a parte de processos **daemon**), engenharia de *software* e infraestrutura de rede pode ser útil para uma melhor compreensão do Fiware e dos conceitos que o permeiam.

A realização deste trabalho mostrou-se uma atividade desafiadora, mas de vital importância, uma vez que explora uma ferramenta poderosa na área de Internet das Coisas. Para além disso, os autores esperam que esse trabalho possa servir como um tutorial para alunos e pesquisadores, dada a escassez desse tipo de material, ainda mais em língua portuguesa. Também por isso, preocuparam-se com uma descrição formal dos conceitos aqui apresentados, ilustraram o funcionamento de todo o sistema e explicitaram todo o código utilizado. O vídeo-demonstração também corrobora para isso, ao exibir o comportamento esperado do sistema para cada instrução.

Referências

Amaral, F. V., Juliani, J. P., and Bettio, R. W. d. (2021). Internet das coisas em bibliotecas: proposta de um sistema para monitoramento de ruído para bibliotecas. *Em Questão*, 28(1):458–483.

Atzori, L., Iera, A., and Morabito, G. (2010). The internet of things: A survey. *Computer Networks*, pages 2787–2805.

Bashari Rad, B., Bhatti, H., and Ahmadi, M. (2017). An introduction to docker and analysis of its performance. *IJCSNS International Journal of Computer Science and Network Security*, 173:8.

Booch, G., Rumbaugh, J., and Jacobson, I. (2000). *UML - Guia Do Usuario*. Campus Editora RJ, 1st edition.

da Cerveja, G. (2019). Guia da Cerveja inovação cervejeira: Conheça tecnologias que estão mudando a cerveja moderna.

de Padua Paula Filho, W. (2019). *Engenharia de Software - Produtos*. LTC, 1st edition.

Forbes, E. (2021). Forbes ambev aposta em "bom ano"para indústria de cerveja no brasil em 2022.

Gershenfeld, N., Krikorian, R., and Cohen, D. (2004). The internet of things. *Scientific American*, 291:76–81.

Guner, A., Kurtel, K., and Celikkan, U. (2017). A message broker based architecture for context aware iot application development. In *2017 International Conference on Computer Science and Engineering (UBMK)*, pages 233–238.

Kovacs, L. (2021). Tecnoblog o que é e para que serve o mongodb?

Moll, V. (2022). Geek Hunter como construir uma aplicação com docker?

Pohl, K. (2012). *Fundamentos da ENgenharia de Requisitos*. Rockynoock, 1st edition.

Prestes, G. and Cordeiro, A. R. (2008). Tecnologia da fabricação de cerveja. *VI Semana de Tecnologia em Alimentos*, 2:1–9.

Sen, S. and Balasubramanian, A. (2018). A highly resilient and scalable broker architecture for iot applications. In *2018 10th International Conference on Communication Systems & Networks (COMSNETS)*, pages 336–341.

Silva, R. B. (2019). Desenvolvimento de um framework para a plataforma fiware. Master's thesis, Universidade Federal de Sergipe, Brasil.

Sommerville, I. (2019). *Engenharia de Sofware*. Pearson Universidades, 10th edition.

Turnbull, J. (2014). *The Docker Book: Containerization is the new virtualization*. James Turnbull, 1st edition.

Vasconcelos, Y. (2021). Revista Pesquisa inovações cervejeiras.

Vazquez, C. E. and Simoes, G. S. (2016). *Engenharia de Requisitos: software orientado ao negócio*. Brasport, 1st edition.

APÊNDICE

Uma outra forma de configurar o ambiente é criar os contêineres um a um e estabelecer uma rede entre eles. O usuário pode fazer isso com os seguintes passos:

- Inicie o contêiner do banco de dados

```
docker pull mongo:4.4
```

- Inicie o contêiner do Orion Context Broker

```
docker pull fiware/orion
```

- Inicie o contêiner do IoT Agent

```
docker pull fiware/iotagent-ul
```

- Crie uma rede para interligar os contêineres

```
docker network create fiware_trabalhoiot
```

- Conecte o contêiner do banco de dados a rede

```
docker run -d --name=mongo-db --network=fiware_trabalho \
  --expose=27017 mongo:4.4   bind_ip_all
```

- Conecte o contêiner do Orion a rede

```
docker run -d --name fiware-orion -h orion --network=
   fiware_default \
  -p 1026:1026  fiware/orion -dbhost mongo-db
```

- Torne o banco de dados persistente

```
docker run --name mongotrabalho -v /home/romulo/mongodata:/
   data/db -d mongo
```

Após este procedimento, basta retornar aos comando exibidos na Seção 4.